

REMARKS

Claims 1–21 are pending in the present application.

Claims 1–21 are rejected.

Claims 1–2, 4, 9–10, 12, 17–18 and 20 were amended herein. Claims 1, 9 and 17 were amended solely for clarity, and not for the purpose of arguing patentability over the cited references. Claims 2, 4, 10, 12, 18 and 20 were amended solely to correct antecedent basis issues created by amendment of their respective base claims.

The title was amended herein for clarity. The specification was amended herein to correct various typographical and grammatical errors, and to conform the Abstract to the proper language and format. No new matter has been added by the amendments to the specification.

Reconsideration of the claims is respectfully requested.

35 U.S.C. § 102 (Anticipation)

Claims 1, 9 and 17 were rejected under 35 U.S.C. § 102 as being anticipated by U.S. Patent No. 6,141,747 to *Witt*. This rejection is respectfully traversed.

A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. MPEP § 2131; *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). Anticipation is only shown where each and every limitation of the claimed invention is found in a single prior art reference. MPEP § 2131; *In re Donohue*, 766 F.2d 531, 534, 226 U.S.P.Q. 619, 621 (Fed. Cir. 1985).

Independent claims 1, 9 and 17 each recite an operand queue in a floating point unit. Such a feature is not shown or suggested by the cited reference. *Witt* teaches that execution cores 40A and 40B may including floating point units, but teaches forwarding logic 62 and rotate/mux circuit 66 as being outside execution core 40A, within load store unit 42. *Witt*, Figure 2, column 11, lines 49–52.

In addition, independent claims 1, 9 and 17 each also recite supplying a first operand from a first floating point processing unit within a floating point unit (e.g., store convert unit 317 within floating point unit 230) to a second floating point processing unit within the floating point unit (e.g., one of load convert units 315a, 315b within floating point unit 230). Such a feature is not shown or suggested by the cited reference. *Witt* does not disclose any details of the construction of execution cores 40A and 40B, particularly the floating point units therein. Moreover, *Witt* teaches that the forwarding logic 62, rotate/mux circuit 66, and other components depicted in Figure 2 are separately implemented for execution cores 40A and 40B. *Witt*, column 11, lines 44–45. *Witt* does not teach or suggest that one execution core 40A or 40B retrieves data from the load/store unit 42 of another execution core 40A or 40B. Accordingly, *Witt* does not teach or suggest this feature.

Therefore, the rejection of claims 1, 9 and 17 under 35 U.S.C. § 102 has been overcome.

35 U.S.C. § 103 (Obviousness)

Claims 2–5, 10–13 and 18–21 were rejected under 35 U.S.C. § 103(a) as being unpatentable over *Witt* in view of U.S. Patent No. 5,721,855 to *Hinton et al.* This rejection is respectfully traversed.

In *ex parte* examination of patent applications, the Patent Office bears the burden of establishing a *prima facie* case of obviousness. MPEP § 2142; *In re Fritch*, 972 F.2d 1260, 1262, 23 U.S.P.Q.2d 1780, 1783 (Fed. Cir. 1992). The initial burden of establishing a *prima facie* basis to deny patentability to a claimed invention is always upon the Patent Office. MPEP § 2142; *In re Oetiker*, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); *In re Piasecki*, 745 F.2d 1468, 1472, 223 U.S.P.Q. 785, 788 (Fed. Cir. 1984). Only when a *prima facie* case of obviousness is established does the burden shift to the applicant to produce evidence of nonobviousness. MPEP § 2142; *In re Oetiker*, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); *In re Rijckaert*, 9 F.3d 1531, 1532, 28 U.S.P.Q.2d 1955, 1956 (Fed. Cir. 1993). If the Patent Office does not produce a *prima facie* case of unpatentability, then without more the applicant is entitled to grant of a patent. *In re Oetiker*, 977 F.2d 1443, 1445, 24 U.S.P.Q.2d 1443, 1444 (Fed. Cir. 1992); *In re Grabiak*, 769 F.2d 729, 733, 226 U.S.P.Q. 870, 873 (Fed. Cir. 1985).

A *prima facie* case of obviousness is established when the teachings of the prior art itself suggest the claimed subject matter to a person of ordinary skill in the art. *In re Bell*, 991 F.2d 781, 783, 26 U.S.P.Q.2d 1529, 1531 (Fed. Cir. 1993). To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed invention and

the reasonable expectation of success must both be found in the prior art, and not based on applicant's disclosure. MPEP § 2142.

As noted above, independent claims 1, 9 and 17, from which the rejected claims depend, recite features not shown or suggested by *Witt*. Such features are also not shown by *Hinton et al*, taken alone or in combination with *Witt*.

Therefore, the rejection of claims 2–5, 10–13 and 18–21 under 35 U.S.C. § 103 has been overcome.

AMENDMENTS WITH MARKINGS TO SHOW CHANGES MADE

The title was amended herein as follows:

OPERAND QUEUE FOR USE IN A FLOATING POINT UNIT TO REDUCE
READ-AFTER-WRITE LATENCY AND METHOD OF OPERATION

The paragraphs bridging page 2, line 22 through page 3, line 20 were amended herein as follows:

Efficiency is particularly important in mathematical calculations, particularly floating point calculations. Some mathematical operations, such as multiplication and division, cause significant delays during program execution. A pipelined floating point unit (FPU) may be particularly susceptible to long delays during the execution of certain sequences of instructions. For example, a floating point “load” instruction may occur in a pipelined FPU immediately after, or shortly after, a floating point store instruction occurs. This is sometimes referred to as a “read-after-write” (RAW) hazard. The write (or store) operation to system memory may have a long latency before the write data is “committed” to system memory by the processor. The read (or load) operation following the write (or store) operation may occur before the write operation is complete and may, therefore, suffer significant delays waiting for

the write operation [is]to complete before the committed data may be read back from memory.

Therefore, there is a need in the art for an improved microprocessor that executes mathematical operations more rapidly. In particular, there is a need for an improved floating point unit that executes floating point operations as rapidly as possible. More particularly, there is a need in the art for a floating point unit that minimizes delays caused by writing data to memory.

The paragraph on page 5 at lines 16–21 of the specification was amended herein as follows:

In a further embodiment of the present invention, the data in the external memory is accessed in groups of N bytes and [wherein] the floating point unit further comprises at least one aligner capable of receiving a first incoming operand that is misaligned with respect to a boundary between a first N byte group and a second N byte group and aligning the first incoming operand.

The paragraphs bridging page 9, line 10 through page 11, line 20 were amended herein as follows:

FIGURE 1 illustrates processing system 10, which includes integrated

microprocessor 100, according to one embodiment of the present invention. Integrated microprocessor 100 comprises central processing unit (CPU) 105, which has dual integer and dual floating point execution units, separate load/store and branch units, and level one (L1) instruction and data caches. Microprocessor 100 also comprises graphics unit 110, system memory controller 115, and level two (L2) cache 120, which is shared by CPU 105 and graphics unit 110. Graphics unit 110, system memory controller 115, and L2 cache 120 may be integrated onto the same die as CPU 105. Bus interface unit 125 couples CPU 105, graphics unit 110, and L2 cache 120 to memory controller 115. Bus interface unit 125 also may be integrated onto the same die as CPU 105.

Integrated memory controller 115 bridges microprocessor 100 to system memory 140, and may provide data compression and/or decompression to reduce bus traffic over external memory bus 145 which preferably, although not exclusively, has a RAMbus™, fast synchronous dynamic random access memory (SDRAM) or other type protocol. Integrated graphics unit 110 provides thin film transistor (TFT), DSTN, red-green-blue (RGB), and other types of video output to drive display 150.

Bus interface unit 125 connects microprocessor 100 through input/output (I/O) interface 130 to PCI bridge 155, which has a conventional peripheral component interconnect (PCI) bus interface on PCI bus 160 to one or more peripherals, such as sound card 162, local area network (LAN) controller 164, and

disk drive 166, among others. Bus interface unit 125 also connects fast serial link 180 and relatively slow I/O port 185 to microprocessor 100 (via I/O interface 130 and PCI bridge 155). Fast serial link 180 may be, for example, an IEEE 1394 bus (i.e., “Firewire”) and/or a universal serial bus (“USB”). I/O port 185 is used to connect peripherals to microprocessor 100, such as keyboard 190 and/or a mouse. In some embodiments, PCI bridge 155 may integrate local bus functions such as sound, disk drive control, modem, network adapter, and the like.

The paragraph on page 12 at lines 8–18 of the specification was amended herein as follows:

In the exemplary embodiment, FPU 230 uses two load buses because the frequency of load operations is twice the frequency of floating point operations. Therefore, in order to achieve an execution rate of one floating point operation per clock, FPU 230 uses two load buses 240. FPU 230 uses one store bus 245 to store results to system memory 140 at commit time. Unlike load operations, where the memory alignment is done in FPU 230, rotating data to put it in memory format is done in data cache 220. The reason for one store bus is that store operations only comprise between 5% and 15% of all floating ~~[pint]~~point instructions, so one bus is sufficient for bandwidth purposes.

The paragraph bridging page 13, line 14 through page 14, line 5 of the specification was amended herein as follows:

FPU 230 receives opcodes (instructions) from instruction decoder[\]/Ucode logic 210. Since the number of bits required to control FPU 230 may be quite large, instruction decoder[\]/Ucode logic 210 does not send FPU 230 a micro-word. Instead, instruction decoder[\]/Ucode logic 210 sends index values to FPU micro-ROM (UROM) 302. The index values are represented by the inputs instruction/microcode (IU) index (0) to instruction[\code]/microcode (IU) index (3). UROM 302 outputs consists of an add/multiply operation and a load store operation that are applied to node exchange (XCH)/register mapping logic and logical-to-physical register file logic 304. XCH/Reg & Mapping and LRF logic 304 computes the physical source and destination addresses in system memory 140 of an operand for each instruction in system memory 140 using register offset values represented by inputs register offset (0) through register offset (3).

The paragraphs bridging page 14, line 18 through page 15, line 15 of the specification were amended herein as follows:

Finally, the data may not have been computed yet. In this final case, the

[dependant]dependent instruction is marked as pending and the PRF location where the data will be deposited is returned. The [dependant]dependent instruction then monitors the result busses and when the result is produced, PRF 330 is read to obtain the data. Once the operation and physical locations of the operands have been generated, the opcodes are loaded into opcode queues 341-344 associated with each functional unit and into a content addressable memory (CAM) which controls the operand valid bits.

There are four major functional units in FPU [239]230. Adder 311 and multiplier 313 perform the majority of the arithmetic. These operations are fully pipelined and have a latency of three clock cycles and a throughput of one clock cycle. FPU 230 uses two load conversion units 315a and 315b to convert load data from a format stored in system memory 140 to the internal format of FPU 230. Load conversion units 315a and 315b receive operands only from operand queue 345. When all pieces of load data in operand queue 345 are valid, one of load conversion units 315a and 315b is scheduled to convert the load data. The opcode in opcode queue 343 indicates how wide the load data is and what format conversion the load data requires.

The paragraph on page 19 at lines 3–11 of the specification was amended herein as follows:

As used herein, “virtual commit” is the process of transferring store data from operand queue 345 into virtual commit buffer 350, as well as storing virtually committed data into any dependent load slots in operand queue 345. The process of virtual commit is performed on a slot-by-slot basis in operand queue 345 and virtual commit buffer 350. However, a virtual commit cycle is only required if a slot has a floating-point store in it. Checkpoints that do not have any floating-point stores also require 1 cycle to virtually commit.

The paragraphs on page 20 at lines 3–21 of the specification were amended herein as follows:

The virtual commit pointer is advanced as quickly as it can be through the slots in virtual commit buffer 350. This means that the virtual commit pointer does not wait for a store to complete for it to advance. Instead, as soon as a checkpoint has been issued to the load/store unit, the virtual commit pointer pulls all stores from operand queue 345 and forwards data from the store operation to any [dependant]dependent read operation. The virtual commit pointer only stops after all stores for the three virtual commit checkpoints have been read.

When a store occurs, the store data is written into operand queue 345 at the address indexed by the store slot:checkpoint value and the CAMs in forwarding array 351 compare the store address with all “forward from” addresses so that all

[dependant]dependent reads will be updated as well. The CAM outputs are used as word lines for operand queue 345 and are also used to mark the [dependant]dependent reads as needing re-execution. Store operations also write into virtual commit buffer 350 at the proper slot:checkpoint value, so that it is not necessary to back up the virtual commit pointer to the slot:checkpoint value where the store occurred.

The paragraph on page 32, at lines 5–18 of the specification (the “ABSTRACT OF THE DISCLOSURE”) was amended herein as follows:

[There is disclosed an operand queue for use in a floating point unit. The]A floating point unit [comprises]includes floating point processing units for executing floating point instructions that write operands to an external memory and for executing floating point instructions that read operands from the external memory. The floating point also [comprises]includes an operand queue for storing a plurality of operands associated with one or more operations being processed in the floating point unit. The operand queue stores a first operand [being] written [to an external memory] by a floating point write instruction executed by a first one of the plurality of floating point processing units and supplies the first operand to a floating point read instruction executed by a second one of the plurality of floating point processing

units [subsequent to the execution of the floating point write instruction]when the first operand is committed or virtually committed.

Claims 1–2, 4, 9–10, 12, 17–18 and 20 were amended herein as follows:

1. (amended) For use in a data processor, a floating point unit comprising:

a plurality of floating point processing units capable of executing floating point instructions that write operands to an external memory and capable of executing floating point instructions that read operands from said external memory; and

an operand queue capable of storing a plurality of operands associated with one or more operations being processed in said floating point unit, wherein said operand queue stores a first operand [being] written [to an external memory] by a floating point write instruction executed by a first one of said plurality of floating point processing units and wherein said operand queue supplies said first operand to a floating point read instruction executed by a second one of said plurality of floating point processing units [subsequent to said execution of said floating point write instruction]when said first operand is committed or virtually committed if said floating point read instruction requires said first operand.

1 2. (amended) The floating point unit as set forth in Claim 1 wherein said floating point unit
2 further comprises a store conversion unit capable of converting operands in said plurality of floating
3 point processing units from an internal format associated with said plurality of floating point
4 processing units to an external format associated with [said]an external memory.

1 4. (amended) The floating point unit as set forth in Claim 1 wherein said floating point unit
2 further comprises a load conversion unit capable of converting incoming operands received from
3 said external memory from an external format associated with [said]an external memory to an
4 internal format associated with said plurality of floating point processing units.

1 9. (amended) A data processor comprising:

2 at least one pipelined integer execution unit;

3 a data cache;

4 an instruction cache; and

5 a floating point unit comprising:

6 a plurality of floating point processing units capable of executing floating
7 point instructions that write operands to an external memory and capable of executing
8 floating point instructions that read operands from said external memory; and

9 an operand queue capable of storing a plurality of operands associated with
10 one or more operations being processed in said floating point unit, wherein said operand
11 queue stores a first operand [being] written [to an external memory] by a floating point write
12 instruction executed by a first one of said plurality of floating point processing units and
13 wherein said operand queue supplies said first operand to a floating point read instruction
14 executed by a second one of said plurality of floating point processing units [subsequent to
15 said execution of said floating point write instruction]when said first operand is committed
16 or virtually committed if said floating point read instruction requires said first operand.

1 10. (amended) The data processor as set forth in Claim 9 wherein said floating point unit
2 further comprises a store conversion unit capable of converting operands in said plurality of floating
3 point processing units from an internal format associated with said plurality of floating point
4 processing units to an external format associated with [said]an external memory.

1 12. (amended) The data processor as set forth in Claim 9 wherein said floating point unit
2 further comprises a load conversion unit capable of converting incoming operands received from
3 said external memory from an external format associated with [said]an external memory to an
4 internal format associated with said plurality of floating point processing units.

1 17. (amended) For use in a floating point unit comprising a plurality of floating point
2 processing units capable of executing floating point instructions that write operands to an external
3 memory and capable of executing floating point instructions that read operands from the external
4 memory, a method of accessing the operands comprising the steps of:

5 storing in an operand queue a first operand [being] written [to the external memory]
6 by a floating point write instruction executed by a first one of the plurality of floating point
7 processing units; and

8 supplying the first operand from the operand queue to a floating point read instruction
9 executed by a second one of the plurality of floating point processing units [subsequent to the
10 execution of the floating point write instruction]when the first operand is committed or virtually
11 committed if the floating point read instruction requires the first operand.

1 18. (amended) The method as set forth in Claim 17 wherein the floating point unit further
2 comprises a store conversion unit capable of converting operands in the plurality of floating point
3 processing units from an internal format associated with the plurality of floating point processing
4 units to an external format associated with [the]an external memory.

1 20. (amended) The method as set forth in Claim 17 wherein the floating point unit further
2 comprises a load conversion unit capable of converting incoming operands received from [the]an
3 external memory from an external format associated with the external memory to an internal format
4 associated with the plurality of floating point processing units.

SUMMARY


If any issues arise, or if the Examiner has any suggestions for expediting allowance of this Application, the Applicant respectfully invites the Examiner to contact the undersigned at the telephone number indicated below or at *dvenglarik@davismunck.com*.

The Commissioner is hereby authorized to charge any additional fees connected with this communication or credit any overpayment to Deposit Account No. 50-0208.

Respectfully submitted,

DAVIS MUNCK, P.C.

Date: 11-27-02



Daniel E. Venglarik
Registration No. 39,409

P.O. Box 800889
Dallas, Texas 75380
(972) 628-3616 (direct dial)
(972) 628-3600 (main number)
(972) 628-3616 (fax)
E-mail: *dvenglarik@davismunck.com*